**California State University, Fresno**

**Lyles College of Engineering**

**Electrical and Computer Engineering Department**

**TECHNICAL REPORT**

**Experiment Title:** Self-Correcting State Machine

**Course Title:** ECE 176 Computer Aided Design

**Date Submitted:** December 9, 2014

**Honor Code Statement:**

**“I have done my own work and have neither given nor received**

**unauthorized assistance on this work.”**

|  |  |
| --- | --- |
| **Prepared By:** |  |
| **Christopher Hays** |  |
|  |  |
| **Signature:** |  |

**INSTRUCTOR SECTION**

**Comments:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**TABLE OF CONTENTS**

**Section:** **Page:**

Title Page ……………………………………………………………………...…. 1.

Table of Contents ……………………………………………………………….... 2.

1. Statement of Objectives ……………………………………………………….. 3.

2. Theoretical Background …………………….…………………………………. 3.

3. Experimental Procedure ………………………………………………………. 3.

3.1 Equipment Used ……………………………………………………... 3.

3.2 Laboratory Procedure ……………………………………………...… 3.

4. Analysis ……………………………………………………………………...... 4.

5. Conclusions ……………………………………………………………………. 6.

6. Appendix ………………………………………………………………………. 7.

**1. STATEMENT OF OBJECTIVES**

The objective of this assignment was to design a self-correcting state machine in Quartus II using the Verilog Hardware Definition Language. The machine was designed to count the sequence 1, 2, 4, 6 and to resume the correct sequence if forced into an unused state; the design also included a clock input, asynchronous reset, and used D Flip Flops. The current state output was sent to a seven-segment display controller to be output to the Altera DE2 development board. The state machine counter was synthesized with the Cyclone II as the target hardware and an RTL simulation was conducted in ModelSim software to verify its operation.

**2. THEORETICAL BACKGROUND**

Self-correcting state machines have states that are unused in the main operation of the machine; these are caused by having leftover possible states after using the minimum number of flip-flops to design the circuit. These leftover states must be given assignments to correct the machine if an unused state is entered into accidentally. In this particular design, 5 states are needed to count the desired sequence, requiring 3 flip-flops (23 = 8). These 3 unused states should not be accessed under normal operation, but noise on a line may force the machine into them. In this design when noise forces the machine into an unused state, the error is corrected in one clock cycle (assuming the noise is gone) and resumes counting at the next number in the sequence.

**3. EXPERIMENTAL PROCEDURE**

**3.1 Equipment Used**

Altera ModelSim Software

Altera Quartus II

**3.2 Laboratory Procedure**

First, the state machine diagram, tables, and equations were derived on paper (Figure 4, Appendix). A new project was created in Quartus II, using the Cyclone II as the targeted hardware, Verilog as the design language, and ModelSim as the simulation environment. A custom D Flip-Flop module was created which had clock, reset, and data inputs as well as a data output. Three of these modules were used to represent the bits of the 5 required states (D2 D1 D0). A seven-segment display controller from a previous project was also included in the project. Next, a main self-correcting state machine module was created, containing the three instantiated flip-flops, an instantiated seven-segment controller, a clock input, line input, reset input, and seven outputs to represent the current count on the development board.

The flip-flop outputs were assigned to the internal wires Y2, Y1, and Y0. These wires, along with the module input, were used to assign the input equations for each flip-flop. These equations ensure the proper flip-flop input depending on the current state and the value of the data input. The input to the seven-segment controller was a function of the current flip-flop outputs and the output of this controller was tied to the main output of the parent module. Using the pin assignment manager, the clock input, reset line, data line, and 7 outputs were all assigned pins corresponding to the DE2 development board (Figure 1). The circuit counts when the data line is logic 1 and a pulse of logic 0 will be used to simulate noise.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| To | Direction | Location | I/O Bank | VREF Group | Fitter Location | Description |
| clk | Input | PIN\_G26 | 5 | B5\_N0 | PIN\_G26 | KEY[0] |
| reset | Input | PIN\_N25 | 5 | B5\_N1 | PIN\_N25 | SWITCH[0] |
| seg\_a | Output | PIN\_AF10 | 8 | B8\_N0 | PIN\_AF10 | SEVENSEGMENT0[0] |
| seg\_b | Output | PIN\_AB12 | 8 | B8\_N0 | PIN\_AB12 | SEVENSEGMENT0[1] |
| seg\_c | Output | PIN\_AC12 | 8 | B8\_N0 | PIN\_AC12 | SEVENSEGMENT0[2] |
| seg\_d | Output | PIN\_AD11 | 8 | B8\_N0 | PIN\_AD11 | SEVENSEGMENT0[3] |
| seg\_e | Output | PIN\_AE11 | 8 | B8\_N0 | PIN\_AE11 | SEVENSEGMENT0[4] |
| seg\_f | Output | PIN\_V14 | 8 | B8\_N0 | PIN\_V14 | SEVENSEGMENT0[5] |
| seg\_g | Output | PIN\_V13 | 8 | B8\_N0 | PIN\_V13 | SEVENSEGMENT0[6] |
| x | Input | PIN\_W26 | 6 | B6\_N1 | PIN\_W26 | KEY[3] |

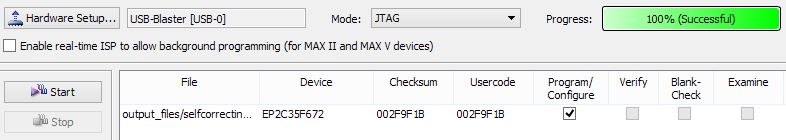


Figure 1: Successful Board Programming

The module was then compiled in order to create a .vo file. Next, analysis and synthesis was done which opened up ModelSim for RTL simulation (Figure 2). A simple testbench file was created that set the initial input values, pulsed the data line to simulate noise, and tested the reset line; this was compiled in ModelSim along with the .vo file, which provided the hardware timing. A clock is simulated by inverting the clk signal every 10 nanoseconds. The code used is in the Appendix.

**4. ANALYSIS**

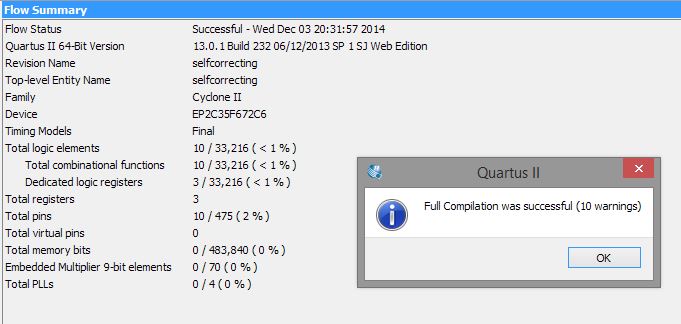


Figure 2: Successful Compilation

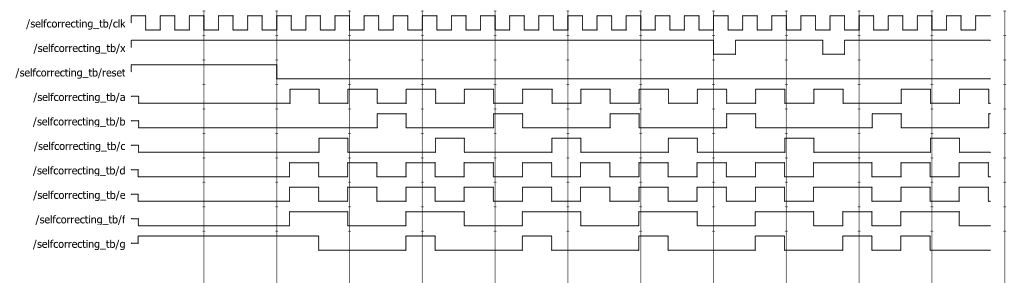


Figure 3: Output Waveforms

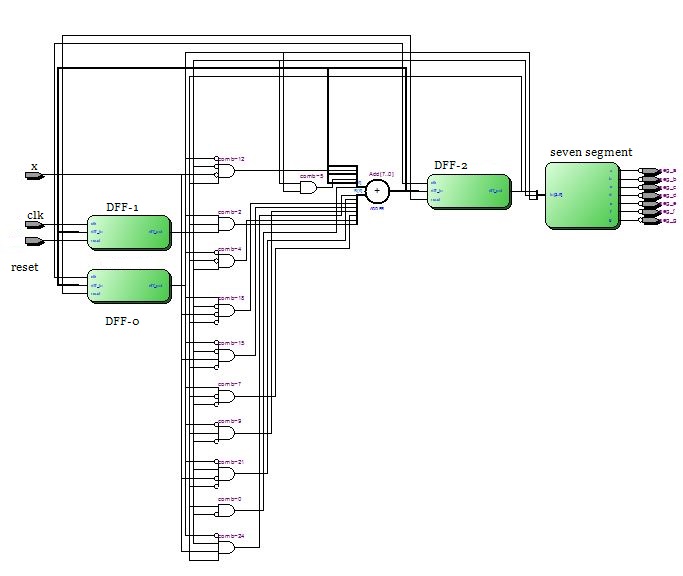


Figure 4: Block Diagram from Quartus II

The simulation begins by setting the initial values of the inputs at time 0 (Figure 3). The data and reset line initialize to logic 1, along with the clock. After 100 nanoseconds the reset line is dropped to logic 0 and the state machine begins counting. The current state is represented on the seven output lines. The output signals here correspond to a seven-segment display that is active low, as on the DE2 development board. The circuit counts three full cycles of 1, 2, 4, 6 then a noise pulse is introduced after the next 2 in the sequence; it occurs exactly at the positive edge of the clock and is over before the next positive edge, so it has no effect on the counting output. The next noise pulse occurs while the machine is in state 4, causing the machine to enter state 7. State 7 causes the machine to proceed to state 6, resuming the counting sequence where it was interrupted. Since the noise pulse is over, the counting continues as desired until the simulation is stopped.

Similar testing was done on the development board. KEY[0] is used as the clock pulse and when pressed, the seven-segment output displays the next number in the sequence of 1, 2, 4, 6. When SWITCH[0] is driven high, the output remains in state 0 (the reset state) even if the clock is pulsed. KEY[3] is used to simulate noise. When KEY[3] is held down and KEY[0] is used to pulse the clock, the output will show the undesired state. The next clock pulse will resume the desired counting cycle. If noise is still present the machine will enter the next undesired state and if noise persists indefinitely then the machine will end up at state 0. The block diagram of the state machine design is shown in Figure 4.

**5. CONCLUSIONS**

The selfcorrecting\_tb.v testbench simulation file and testing on the development board demonstrated that the design of the self-correcting state machine was correct and working. The states counted as expected as long as the input line remained at logic 1. Pulsing the input line to represent noise would cause the counting to be interrupted, but it would resume after one clock cycle. Testing on the development board confirmed this behavior. The .vo file generated by Quartus II helped identify that the propagation delay on the Cyclone II is about 4 nanoseconds in this case. Designs such as this can help to ensure proper operation of a circuit that has been interrupted and also avoid a deadlocked state that the circuit can never recover from.

**6. APPENDIX**

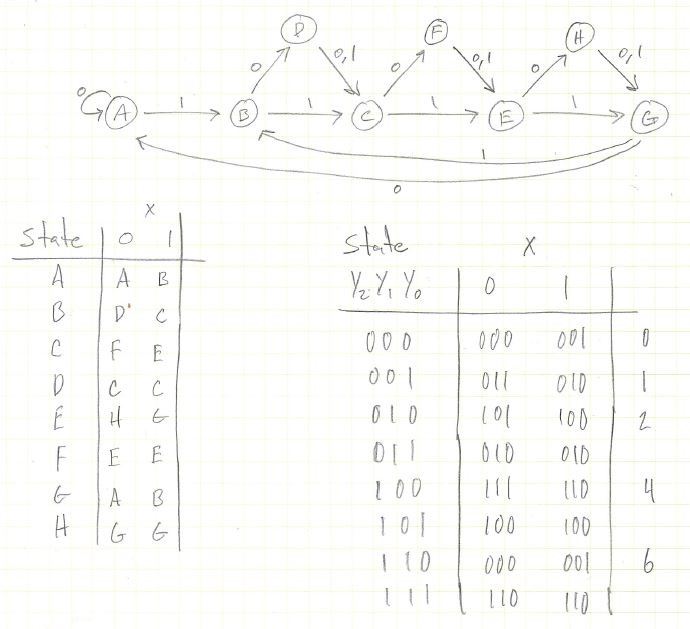


Figure 5: State Diagram

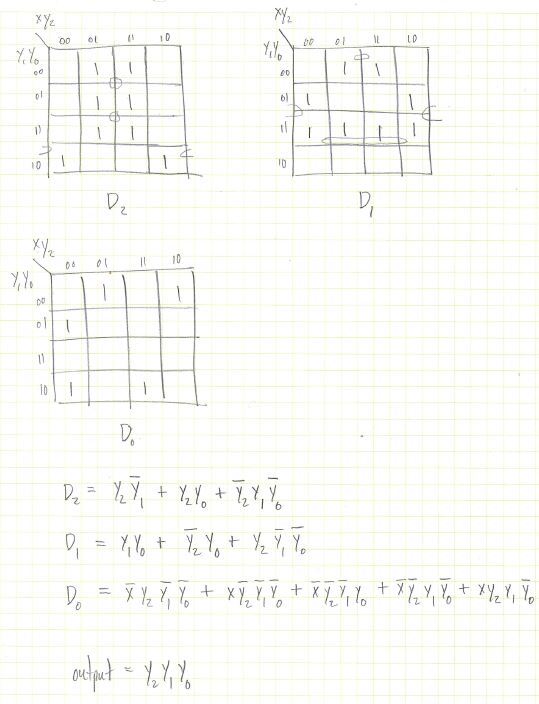


Figure 6: K-Maps and Equations

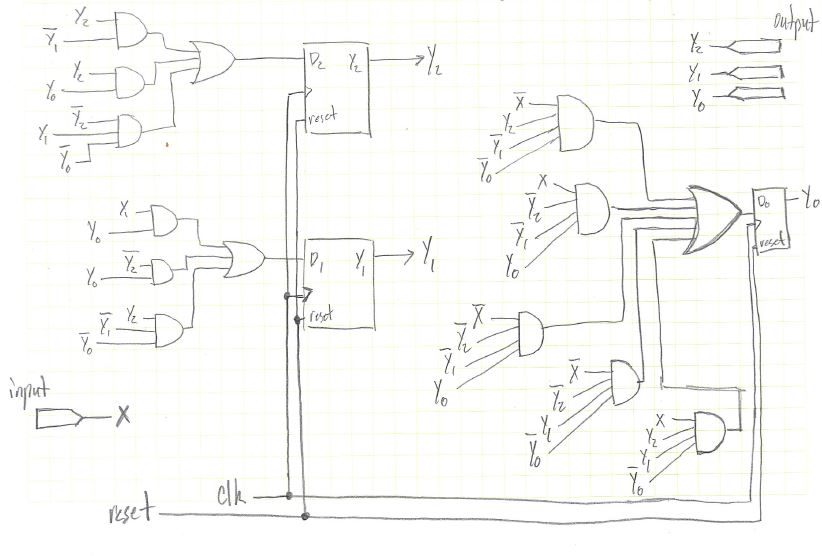


Figure 7: State Machine Circuit

// Christopher Hays

// ECE 176

// Assignment 12

// Self Correcting State Machine

// selfcorrecting.v

module selfcorrecting(input clk, x, reset, output reg seg\_a, seg\_b, seg\_c, seg\_d, seg\_e, seg\_f, seg\_g);

// current flip-flop outputs

wire Y2, Y1, Y0;

wire a\_, b\_, c\_, d\_, e\_, f\_, g\_;

// state equations assigned to the flip-flop inputs

wire D2 = (Y2 & ~Y1) + (Y2 & Y1 & Y0) + (~Y2 & Y1 & ~Y0),

D1 = (Y1 & Y0) + (~Y2 & ~Y1 & Y0) + (Y2 & ~Y1 & ~Y0),

D0 = (~x & Y2 & ~Y1 & ~Y0) + (x & ~Y2 & ~Y1 & ~Y0) + (~x & ~Y2 & ~Y1 & Y0) + (~x & ~Y2 & Y1 & ~Y0) + (x & Y2 & Y1 & ~Y0);

// most significant bit of the state

dff\_custom DFF2(.clk(clk),

.dff\_in(D2),

.dff\_out(Y2),

.reset(reset)

);

// middle bit of the state

dff\_custom DFF1(.clk(clk),

.dff\_in(D1),

.dff\_out(Y1),

.reset(reset)

);

// least significant bit of the state

dff\_custom DFF0(.clk(clk),

.dff\_in(D0),

.dff\_out(Y0),

.reset(reset)

);

// the seven segment controller

sevensegment S1 (.in({Y2, Y1, Y0}),

.a(a\_),

.b(b\_),

.c(c\_),

.d(d\_),

.e(e\_),

.f(f\_),

.g(g\_)

);

// output assignments, seven segment displays are active low so complement

always begin

seg\_a <= ~a\_;

seg\_b <= ~b\_;

seg\_c <= ~c\_;

seg\_d <= ~d\_;

seg\_e <= ~e\_;

seg\_f <= ~f\_;

seg\_g <= ~g\_;

end

endmodule

// Christopher Hays

// ECE 176

// Assignment 12

// testbench

// selfcorrecting\_tb.v

`timescale 1ns/100ps

module selfcorrecting\_tb();

reg clk, x, reset;

wire a, b, c, d, e, f, g;

selfcorrecting U1 (.clk(clk),

.seg\_a(a),

.seg\_b(b),

.seg\_c(c),

.seg\_d(d),

.seg\_e(e),

.seg\_f(f),

.seg\_g(g),

.reset(reset),

.x(x)

);

initial begin

#0 clk = 1;

#0 x = 1;

#0 reset = 1;

#100 reset = 0;

#300 x = 0;

#15 x = 1;

#60 x = 0;

#15 x = 1;

#100 $stop;

end

always

#10 clk = ~clk;

endmodule

// Christopher Hays

// ECE 176

// Assignment 12

// D Flip Flop

// dff.v

module dff\_custom(input clk, dff\_in, reset, output reg dff\_out);

always @(posedge clk, posedge reset)begin

if(reset)

dff\_out = 0;

else

dff\_out = dff\_in;

end

endmodule

// Christopher Hays

// ECE 176 Fall 14

// Assignment 12

// Seven segment display controller

module sevensegment(in, a, b, c, d, e, f, g);

input [2:0] in;

output a, b, c, d, e, f, g;

reg [6:0] out;

always @(\*)begin

case (in)

3'b000 : out = 7'b1111110;

3'b001 : out = 7'b0110000;

3'b010 : out = 7'b1101101;

3'b011 : out = 7'b1111001;

3'b100 : out = 7'b0110011;

3'b101 : out = 7'b1011011;

3'b110 : out = 7'b1011111;

3'b111 : out = 7'b1110000;

default : out = 7'b1001111;

endcase

end

assign {a, b, c, d, e, f, g} = out;

endmodule